Ryerson University

Department of Electrical, Computer & Biomedical Engineering
Faculty of Engineering & Architectural Science

Breathing Monitor for Real Time Rate Detection

by

Alexandre Gagné, Danielle Iaboni, and Justine Co

Biomedical Engineering Capstone Design Project

Ryerson University, 2019

Introduction

In recent years, there has been an influx of research dedicated to the remote, non-invasive detection of vital signs in both the clinical and biomedical fields. In many clinical diagnostic and monitoring systems, the respiratory rate is an important parameter of interest. The respiratory rate is defined as the rate at which breathing occurs, usually measured in breaths per minute. The act of breathing is an automatic, seemingly effortless task involving the expansion and contraction of the chest cage [1]. Typical respiration has a fairly constant rate, leading to a respiratory rhythm [1]. For a healthy human adult, the typical respiration rate is 12 to 20 breaths per minute [2]. However, abnormalities may occur in the rate of respiration, leading to various breathing pathologies.

One common breathing rate pathology is apnea. Apnea is defined as the cessation or stopping of breathing [3]. A person who has been diagnosed with apnea will exhibit no movement of the muscles involved in inhalation and the volume of their lungs will remain unchanged [3]. It is estimated that approximately 22 million people in North America suffer from apnea and sleep induced apnea [3].

In addition to apnea, tachypnea is another very common breathing pathology. Tachypnea is characterized by an increase in the breathing rate of humans [4]. People who suffer from tachypnea generally exhibit a breathing rate upwards of 20 breaths per minute, and this can be brought on by both physiological or pathological causes [4]. The physiological act of exercise may induce tachypnea, but it can also be present in those who exhibit pathological diseases such as pneumonia, congestive heart failure and asthma [4].

Another common breathing rate pathology present in many people around is bradypnea. Bradypnea is classified as an abnormally slow rate of breathing [4]. Adults who experience bradypnea generally exhibit a breathing rate of less than 12 breaths per minute [4]. Bradypnea is commonly caused by a variety of pathological issues such as heart degeneration, high blood pressure and congenital heart defects [4].

In order to diagnose a breathing pathology, the rate of respiration must be measured. Generally, the measurement of respiration is achieved by contact methods [5]. These methods include the use nasal thermocouples, oximetry probes, and electrocardiography (ECG) to measure the respiration rate [5]. However, most of these methods are inconvenient, and constrain the patient to a clinical setting, proving to be expensive and inflexible [5]. Thus, there is an increasing need for the use and implementation of remote breathing monitors for the measurement of respiration rate. The implementation of a real-time monitoring system for respiration rate plays an important role in both diagnosing and monitoring a variety of disorders.

The application of motion sensors and accelerometers to detect small movements of the chest wall during respiration is an area of interest [5]. The main objective of this paper is to provide a new app-based tool for monitoring and analyzing respiration rates with an accelerometer sensor. Previous approaches are primarily based on the use and implementation of offline signal processing [5]. In our proposed system, the accelerometer data is transmitted via Bluetooth to a microprocessor, which then transmits the data to a cellular device to be stored and processed. A

physician or caregiver would be able to track the patients regardless of their location with respect to the patient.

Acknowledgements

We wish to express our sincerest gratitude to Professor Dr. B. Ghali for her guidance and encouragement in carrying out this project work. We would also like to thank Mr. J. Koch and other staff members of Ryerson University who rendered their help during the period of our project work.

Certification of Authorship

Submitted to: Dr. B. Ghali

Student's Names: Alexandre Gagné, Danielle Iaboni, and Justine Co

Date of Submission: April 12th, 2019

Purpose and Title of submission: Biomedical Engineering Capstone Design Project, Breathing Monitor for Real Time Rate Detection

Certification of Authorship: We hereby certify that we are the authors of this document and that any assistance received in its preparation is fully acknowledged and disclosed in the document. We have also cited all sources from which we obtained data, ideas, or words that are copied directly or paraphrased in the document. Sources are properly credited according to accepted standards for professional publications. We also certify that this paper was prepared by us for this purpose.

Students's Signatures: A. Gagne, D. Iaboni, J.Co

Table of Contents

Abstract

In many scenarios, the continuous monitoring of a patient’s breathing signal has been proven to be costly and ineffective due to certain methods of approach. There are many different conventional contact methods that can be used to measure and monitor the breathing rate of a patient. They include, but are not limited to, spirometers and nasal thermocouples. Albeit their accuracy, these methods have proven to be expensive and are not mobile devices, as they must be used in a hospital setting. In our work, we present a real-time wireless wearable respiration monitoring device that would allow for continuous monitoring from a home setting. This would allow for the continuation of treatment and diagnosis outside of a hospital setting. Our device prototype utilizes a 3-axis MEMS accelerometer, Bluetooth Low Energy (BLE) data transfer, and signal processing within MATLAB to monitor and measure the breathing rate. The obtained results show that our device is feasible and can be used as a home-based alternative for measuring and monitoring both healthy patients and patients who suffer from a breathing abnormality.

Keywords: Respiration Rate, Accelerometer, Bluetooth, Wearable Device, Real-Time, Signal Processing

Objectives

In this section, the objectives of our design project will be listed, highlighting the key components and motivation for our work.

- Explore and document previous related works for respiration monitoring
- Design and build a wireless respiration monitor to meet the needs of the supervisor
    - Easy to use
    - Simple circuitry
    - Real time signal processing
    - Robustness
- Compare alternative design approaches
- Document the design and implementation procedures

This section outlined the objectives of our design project, including all aspects of research and implementation. In the next section, background information will be provided on previous works that were applicable to our research.

Background

In this section, various methods will be explored for the monitoring and calculation of respiratory rate. All aspects, including hardware, microcontroller and software components, will be fully reviewed and discussed.

6.1 Hardware

Methods for deriving respiratory rate include conventional processes such as spirometry, nasal thermocouples and whole-body plethysmography [5-7]. These processes require either face-masks, mouthpieces or thermocouples that are uncomfortable to wear for extended periods of time [8]. Other methods are limited to stationary positions. A Doppler radar, image sequence analysis or thermal cameras do not allow the user to be in motion [5,9]. Small movements of the chest have been the focus for acquiring respiratory rate [6-8,9-11].

The thoracic or abdominal expansion can be measured using several types of sensors. Studies have used optical fibre pressure sensors, accelerometers, inductance plethysmography, impedance pneumography and strain gauges. In [8], an optical fibre-based monitoring system to measure the respiration rate is used. The benefit of using an optical fibre is that it is insensitive to electromagnetic fields, water and corrosion and is suitable for long-term monitoring. Respiratory rate is measured by means of signal intensity changes. A red LED is used to emit light while a photodiode, encased in a connector, detects the remaining light at the other end of the fibre. The waveguide reacts to applied pressure with a change in light intensity [8]. The use of inertial sensors, or accelerometers, have been applied to detect small movements of the chest wall that occurs during the expansion and contraction of the lungs. Trials have shown that an accelerometer-based monitoring system can produce results that closely match measurements of nasal cannula pressure. [10] used a single 3-axis MEMS accelerometer to derive the respiratory signal from the suprasternal notch. Testing of the device required that the subjects emulated several types of breathing patterns in a lying position. Based on the fixed sensor position, the analysis of the data was only performed on the z-axis of the accelerometer [10]. [6] also used accelerometers to detect the acceleration and angular changes during respiration. However, they used two accelerometers to measure respiration rate while standing, walking and running.

Two accelerometers, centered at the front and back of the torso, were used to eliminate translational movement that does not belong to respiration. A differential measurement between the two accelerometers was used to obtain the respiration rate. Both accelerometers used were 3-axis, MEMS and had high sensitivities. The inclination values were used to align the front and rear accelerometer coordinates. Data processing only involved the y-axis, which is perpendicular to the chest [6]. Inductive plethysmography also generates voltage changes from movement of the rib cage and abdominal area. Sensors are used to measure differences in cross-sectional area of the rib cage and abdominal compartments. A low alternating current, with a high frequency, is produced by an electrical oscillator circuit and passed through arrays of sinusoidally arranged wires around the chest. Movement in the chest area generate magnetic fields which can be measured as voltage changes over time. Impedance pneumography also provides qualitative measurements of chest wall movement. Two surface electrodes record the thoracic movements. A low amplitude alternating current is passed between the two electrodes at a high frequency. The voltage drop across the electrodes is computed as impedance [7]. Lastly, [11] recorded the expansion and contraction of the thoracic region. A respiratory sensor based on a reflective object sensor measured the displacement of the chest. The gauge consists of an immovable end and a displaceable part that slides in and out. An infrared LED bounces light off the object and into a phototransistor. The level of light depends on the distance of the object [11].

6.2 Microcontroller

In order to find a microcontroller suitable for this project, a variety of options were critically analyzed. With more research, microcontroller choices were continuously changing. For example, a PIC microcontroller was first looked at since there is familiar and previous knowledge on it from past projects, specifically the PIC32. The IDE program MPLabX that is compatible with PIC microcontrollers was also understood from these earlier projects. Cespedes [11] uses PIC microcontrollers in their respiratory monitoring system. The microcontrollers used were the PIC16F88 and the PIC16F877. This design includes a respiratory belt unit with its own microcontroller, the PIC16F88, and a master microcontroller in the central monitoring system, the PIC16F877. The data transfer between belt microcontroller and the master microcontroller can be transferred through a serial port connected by a wire, or wirelessly through a decoder, encoder, and antenna pin. This design using two PIC microcontrollers is complex for wireless data transfer and is not an ideal option.

The second option is to incorporate a Raspberry Pi, that is readily available, with Bluetooth connectivity to transfer the data. It is possible to connect the Raspberry Pi to a PIC microcontroller, however after more research, it was shown by Vertens et al. [6] that a microprocessor can simply be connected to a Bluetooth chip. The microprocessor used in their respiratory monitor was not a PIC microcontroller, rather a STM32F4 microprocessor with a Bluetooth chip on board with Bluetooth 4.0 connection. Another portable wireless respiratory monitor by Cao, Zhu, & Que [12] also uses a BC417 Bluetooth chip connected to their microcontroller. This was the most feasible option for this proposed design project. After more researching, a microcontroller with built-in Bluetooth Low Energy (BLE) was discovered in Fekr et al.’s work [10]. Their respiration rate monitor uses the CC2541, however not much information was found in terms of implementing the microcontroller with an IDE software. More information was discovered on the ESP32 microcontroller with its compatibility with Arduino IDE and how to set up its BLE function. Furthermore, the ESP32 has a Wi-Fi connectivity ability that can be utilized in case the BLE function does not work as expected. With more information on the ESP32 and the additional wireless data transfer capability, the overall superior choice over the CC2541.

The ESP32 is featured in many articles, including Kodali and Mahesh’s [13] smart emergency response system. Their system uses three different kinds of sensors connected to the ESP32, including a 3-axis accelerometer, a pulse sensor, and a sensor that can detect the ambient temperature and humidity [13]. This is reassuring to the proposed design which only uses a 3-axis accelerometer. A low-cost photovoltaic system monitor also uses the ESP32 and presents the use of an SD card connected to the SPI pins in order to store text file data [14]. This can be an option in the case where BLE or Wi-Fi does not function properly or can be an addition feature for patients to track their respiratory history.

6.3 Software

Prior to designing the software component of our device, relevant literature was reviewed in order to get a better understanding of the algorithms that have been used in previous works. J. Vertens et al. [6] implement an adaptive filter to measure and calculate the breathing rate of a patient. In order to acquire a signal that can be analyzed, J. Vertens et al. proposed a variety of preprocessing filters. They used an IIR Butterworth bandpass filter with a frequency range between 0.1 and 0.8 Hz [6]. This frequency range for the filter is an appropriate choice based on the data that is being collected from the accelerometer. The data being collected from the accelerometer is a combination of static and dynamic phases of the chest [6]. The frequency band for respiration is commonly between 0.1 and 0.8 Hz, so the design of a pass band filter using this frequency range is advantageous for us in removing the unwanted frequency components related to the dynamic phase of chest movement. However, J. Vertens et al. showed that despite this filtering, the noise was still too high, and decided to implement an adaptive filter [6]. The adaptive filter works in such a way that it adjusts the low and high frequency cut off values based on the maximum power frequency of the signal and a bandwidth frequency based on the activity of the person [6]. The new borders for the filter are then calculated based on the following equation:

$$F_h = f_{max} + f_{bw} \quad f_l = f_{max} - f_{bw} \text{ [6].} \qquad (1)$$

They established that the bandwidth frequency, f_{bw}, would be sent to 0.25 Hz for still activities and 0.50 Hz for sport activities. The use of an adaptive filter is advantageous as it is set in such a way that variance in respiration during different activities is taken into consideration. For peak detection, and calculation of the breathing rate, J. Vertens et al. implemented an embedded peak detection algorithm that would detect the maxima in 18 seconds worth of data [6]. The algorithm checks for peaks in data by looking at every point and doing a simple comparison with the neighboring points [6]. Finally, the breathing rate is calculated by taking the time between the peaks and dividing it by the sampling rate [6]. This is shown in the equation below:

$$T_d = |P_i - P_j| / F_s \text{ [6].} \qquad (2)$$

A similar study published by A. Fekr et al. [16] was also examined to gain more insight on data processing algorithms for the calculation of breathing rate. A. Fekr et al also used an accelerometer to collect the chest movement during respiration [16]. The data processing phase involved passing the collected data from the accelerometer through a 10th order Butterworth low pass filter with a cutoff frequency of 1 Hz [7]. The filtered signal is then displayed as the respiration waveform. From the respiration waveform, a peak detection algorithm is employed [7]. The algorithm defines a customized threshold in order to determine whether the data point of interest is significantly larger than the neighboring data [7]. When the peaks, denoted as *B,* are found, the peak intervals are then calculated using the equation below:

$$y(k) = \sum_{i=1}^{k} |B(i) - B_{ave}| \text{ [16].} \qquad (3)$$

The final paper analyzed for the purpose of understanding various algorithms was presented by P.D Hung et al. [17]. The proposed system mentioned in this paper also makes use of an accelerometer to gather data from chest movement during breathing [17]. When preprocessing the data, the collected signal is normalized and passed through a constant band-pass filter [17]. P. D Hung et al., similarly to J. Vertens et al., also noticed that there is a significant amount of noise that remains in the signal after only implementing a band pass filter. In order to combat this noise, P.D Hung also implement an adaptive filter based on the maximal energy of the signal [17]. The algorithm works by windowing the data into 1-minute segments, and then calculating the maximum frequency, f_o, present in that window between a range of 0.1 and 1 Hz [17]. The use of this frequency range is similar to that used in previous works, as it has been recorded through research that the average respiration peaks be minute is observably between 6 and 60 beats [17]. The data is then filtered using a 4th order Butterworth filter [17], like that mentioned in the paper published by J. Vertens et al. The bandwidth filter boundaries are then conditioned by the following equation:

$$f_1=\max(0.1,f_o-0.4)\text{Hz}\ f_2=f_o+0.4\text{Hz [17].} \qquad (4)$$

The use of this adaptive filter accounts for various movements not associated with the process of breathing. Finally, P.D Hung et al. calculated the breathing rate by implementing manual graphical methods. The use of an adaptive filter would be feasible for our device, however, due to the nature of real time processing, graphical methods of calculating the breathing rate would not be feasible.

This section highlighted the various methods that have been used in previous works, and their relevance to our work in developing the breathing monitor. In the next section, theory and design, various aspects of our project will be presented.

Theory and Design

In this section, the various theory and design aspects of our respiratory monitored will be explained in further detail. This section describes the various hardware, microcontroller and software aspects, and the reason behind the choices that were made.

7.1 Hardware

Most of the literature that was reviewed focused on measuring the small motions of the chest to determine respiratory rate. While this is not a direct measurement of respiratory rate, it allows for a more minimalist design that is comfortable to wear for longer periods of time. We decided to use a single accelerometer to estimate and measure the breathing rate. Accelerometers sense linear acceleration along each of its axes [18]. They do not require electrical oscillator circuits, do not pass electricity through the individual and can record acceleration along multiple axes. The LIS344ALH is a 3-axis MEMS inertial sensor from STMicroelectronics was selected based on its high sensitivity, scale size and low power consumption [19]. High sensitivity is required because the motion of the chest expanding and contracting is small. The acceleration values have a small range, using a smaller scale improves the sensitivity [6]. Having low power consumption allows the device to have a longer on-time.

The circuit design connects the appropriate pins of the sensor to the microcontroller while taking into account the specifications from the data sheets. Rather than using an additional power source, the sensor is powered by the microcontroller. The recommended supply voltage for the accelerometer is 3.3 V, which is also the output of the 3V pin on the microcontroller [19,20]. The input voltage is regulated by the microcontroller at 3.3 V. The sensor consumes little power, drawing only 680 μA of current. External capacitors adjust the bandwidth of the sensor along each of its axes. The cut-off frequency is determined from,

$$ft=1.45\ FCload(x,y,z)[19]. \quad (6)$$

[6] and [10] did not use hardware filtering, therefore we opted to use a bandwidth of 50 Hz. Three 30 nF external capacitors set the bandwidth of the axis's outputs. The axes of the sensor are set to have a scale range of ±2 g. As mentioned earlier, the chest motion is very small. Only a small range is required in the positive and negative directions [19]. Each of the three sensor outputs is connected to analog input pins of the microcontroller. Digital signals control the full-scale selection, self-test and power down pins of the sensor. Each pin is connected to digital output pins of the microcontroller [19,20,21]. These pins are connected to the microcontroller for potential design changes in the code. Each of the pins is left connected to ground for our purposes. The circuit will be printed onto a board. The printed circuit board (PCB) has through holes for the components. To avoid surface mount soldering, we will use an evaluation board version, STEVAL-MKI015V1, of the sensor. The sensor uses a set of 12-pin headers spaced 0.6 inches apart [21]. Through holes on the bottom board line up with the microcontroller pins. The microcontroller board, sensor evaluation board and the external capacitors were soldered onto the PCB. The traces are located on the bottom of the PCB and connect the sensor to the microcontroller.

To measure the breathing motion, the sensor needs to be attached to the chest. From the literature review, there was one similarity between all the designs. A strap that could be wrapped around the torso was used to secure the device [6-8,10,11]. For the case, we drew similarities from the designs used in [6,10]. Autodesk Fusion 360 is a 3D modelling software that you can use to print your designs. The case was made to house the PCB. The case lid is designed to have a snap-fit. A chamfered groove protrudes the case lid and snaps into a slot on the bottom cover. This avoided the need for screws and makes opening the case quick. The case dimensions were reasonable small, relative to the size of the microcontroller, at 4.7 x 8.5 x 3.2 cm. Screw mounts raise the board off the bottom cover, allowing for the battery slide underneath. Velcro holds the 3.7 V, 2000 mAh lithium-ion battery in place. There is a slight gap surrounding the circuit board so that the wire can pass between the battery and microcontroller. The PCB was mounted using four self-threading screws. They were 1/4” long and 0.11” in diameter.

7.2 Microcontroller

The ESP32 peripherals are shown in the appendix as Figure 5. The ESP32 pins that are the most important to make note of are the 18 ADC channels, 2 DAC pins, 2 SPI interfaces, 3 UART interfaces, 10 capacitive sensing GPIOs, and 16 PWM output channels [15]. The input-only pins are the GPIO 34-39 pins as these pins do not have internal pull-up or pull-down resistors. The ADC pins are GPIO 32-39, 0, 2, 4, 12-15, and 25-27. These ADC pins are ideal input pins for the accelerometer sensor since the output voltage of the accelerometer is an analog signal and conversion to a digital signal is necessary for signal processing. The maximum rating voltage is 3.6V and the minimum is -0.3V [15]. From the accelerometer output, low voltage values are expected and should not exceed the 3.6V. If the accelerometer output is close to this value, a simple solution is to implement an external pull-down resistor before the input pin. Many IDE software programs are compatible with the ESP32, however Arduino IDE is chosen.

A skeleton code is shown in the appendix in which some parts are noteworthy. For example, setting up the BLE component of the microcontroller is crucial in the proposed design. Three different types of BLE libraries must be included at the beginning. The GPIO36 pin is arbitrarily chosen as an input pin for the accelerometer output since it is an ADC CH0 pin. The value of this input pin is initialized to ANALOG_VALUE1=0. The value is read in the loop as ANALOG_VALUE1=analogRead(ANALOG_PIN1). The BLE device is initialized with its name that will be identified by other Bluetooth clients. The microcontroller is then set as a BLE server where data will be transferred from the microcontroller to another Bluetooth device. A characteristic UUID and service UUID are also defined. A UUID, short for Universal Unique Identifier, is a 128-bit number used to identify an object or information in computer systems. In this case, it identifies a particular service provided by the Bluetooth device. The properties that the microcontroller will have are to write, read and notify. There is also an advertising function that allows the microcontroller BLE to be shown to other Bluetooth devices and can exchange data. Writing to the Bluetooth client is shown in the code by PROPERTY_WRITE, and notifies the smartphone using pCharacteristic->notify().

From the data sheets of the ESP32, it is important to note that the relationship for ADC is only considered linear but is not perfectly linear [15]. This can cause discrepancies in values especially since values from the accelerometer are low and changes within decimal points can cause noticeable change. Due to this, our solution was to calibrate the voltage by multiplying it by the 3.6V maximum voltage the microcontroller can withstand and dividing this by the maximum expected output value of the accelerometer.

7.3 Software

Prior to designing the software algorithm, previous research was conducted in order to get a better understanding of the respiratory signal. The physiological aspects of the respiratory signal and pathological respiratory signals were taken into consideration when designing the algorithm. The first part of the algorithm involves preprocessing the signal that is transmitted by the microcontroller. The respiratory signal commonly resides in a frequency range from 0.1 to 0.8 Hz [6]. Based on this, a moving average filter will be used to condition the signal. The moving average filter is similar to that low pass filter, having a finite impulse response [6]. The filter takes a given number of samples at a time and takes the average of those samples to produce a single output point.

After the signal is filtered and smoothed, it will be windowed and buffered for analysis. The most common window size used in previous literature is a size of 18 s worth of data [6]. A window of this size will allow for the proper calculation of the respiratory rate [6]. A window of this size will introduce a slight delay in the real time display of the breathing rate, but it is an acceptable delay due to the nature of real time processing [6]. After the data is windowed, it would be put into a buffer for analysis [6]. The buffer would store the data from the 18 s window, and then be used for analysis [6]. After the processing and segmentation of data, the peak to peak detection algorithm would be put into place. The peak to peak algorithm that will be used is like that used by J. Vertens et al [6]. The algorithm that will be used will check for peaks in the windowed data, and then store the peaks in a buffer [6].

The algorithm to determine the peaks was dependent on the quality of the signal. Due to the signal's clean and smooth nature, a simple peak finding function was used. In order to modify the peak finding function, various parameters were set. These parameters included the minimum peak distance and the minimum peak width. These values were set to 15 and 100 data points respectively. Once the peaks were found, the breathing rate was be calculated using the equation presented by J. Vertens et al., shown in equation 2 [6]. Once the breathing rate is found, it can be classified as normal or abnormal. If abnormal, it can be further classified into the 3 types of pathologies mentioned in the introduction section. These pathologies include Bradypnea, Tachypnea and Apnea. If the breathing rate is found to be greater than 20 breaths per minute, the classification will be Tachypnea [4]. If the breathing rate is calculated to be less than 12 breaths per minute, the classification will be Bradypnea [4]. Finally, if no breathing rate is detected, the classification will be Apnea [4]. The flow chart for the signal processing is shown in the appendix.

All the signal processing and classification procedures took place in the MATLAB environment. The data was collected by Bluetooth from the microcontroller and opened in the MATLAB environment. In MATLAB, the respiration signal collected from the accelerometer was displayed in real time, and the breathing rate was computed using the methodology mentioned above.

This section further explained the various theory and design elements that were implemented for our device. In the next section, alternative design methods will be explored, and comparisons will be provided.

Alternative Designs

This section highlights various alternative designs that have been implemented in previous works. This section will focus on all components of the design, including hardware, microcontroller and software alternatives.

The alternative designs include:

8.1 Hardware

One drawback of using an accelerometer is that it is difficult to separate breathing movement from other motion due to acceleration. It has been suggested that this noise can be easily removed because of its high frequency, but [6] found that it wasn't the case. Other efforts to produce better quality respiration signals used a combination of sensors such as accelerometers and gyroscopes. [6] used two accelerometers, one positioned on the chest and one on the back, to isolate respiratory motion. This required a great deal of calibration, since the axis of the two accelerometers had to be aligned. *Results of 6,10*

Using multiple sensors has yet to produce a resulting respiratory signal that is significantly greater than a single accelerometer, given that the user is in a stationary position [6,10]. With no definite solution to removing other motion from the respiratory signal, we focused solely on the breathing patterns of a stationary user.

Our choice of accelerometer was based off the accelerometer sensitivities, presented in [6,10]. The output of our sensor, however, is analog rather than digital. The accelerometer used in [10] provides a digital output using an onboard 12-bit ADC. In our design, we left the conversion to digital for the microcontroller.

8.2 Microcontroller

Our design used an Arduino microcontroller to convert the analog accelerometer data to digital and then send it using Bluetooth. [10] removed the need for the conversion using a microcontroller and simply uses a BLE module to transmit data. The power consumption is nearly 5 times lower than our Arduino. The ESP32 core is power hungry and uses around 130 mA, giving us a battery life of approximately 15 hours [15]. These alternative design considerations directly affect the size of the device. Using a Bluetooth module reduces the size of the circuit and battery. We left BLE for future considerations because MATLAB is unable to communicate with BLE devices.

Some microcontrollers, such as the STM32F4 used in [6], have better digital signal processing capabilities. The ARM core allows for fast arithmetic calculations. Using a digital signal controller could eliminate the need to do any processing outside of the device. Our choice of microcontroller gave us a bit more versatility (ie. Bluetooth Classic, BLE, Wifi) and has several publicly made filter libraries. Signal processing was left to MATLAB, the software component of our design.

8.3 Software

The digital sensor data can be processed and displayed using different methods. It is apparent that the focus of other designs was portability of the device. First, we saw in [6], that the signal was first sent to a cell phone before being sent to the cloud to be processed and stored. This is ideal since users would typically have a phone on their persons. This allows for the greatest range. Similar to our design, the design used by [10] has a constrained range because the data is transmitted using BLE to be processed on a computer. For our application, Bluetooth is adequate since the user is not moving. Future improvements to our design would be to increase the range of the device.

Methods for filtering the accelerometer data varied between designs. The powerful signal processing chip used in [10] implemented an adaptive bandpass filter. The max power from a Fast Fourier Transform (FFT) was used to build a new Butterworth bandpass filter. An adaptive filter was chosen because their design also evaluated subjects who were in motion. [6] used a 10th order Butterworth lowpass filter with a cut-off frequency of 1 Hz. After some testing, we found that our method of a moving average filter was able to remove the high frequency noise.

The algorithms used for peak detection were relatively similar between designs. [6] used a customized threshold to determine if the peak was larger than the localized data. [10] used peak prominence to determine if a peak was found. The classification of patterns done by [6] used an algorithm that used auto-correlation properties of the signals. The algorithm was able to classify many different patterns, including Biot’s, Cheyn-Stokes and Kussmaul.

Material/Component Cost

This section highlights and outlines the various materials that were purchased for the implementation of our design, and the individual unit costs. The total cost is also documented.

Material	Unit Cost ($)	Quantity Purchased	Total ($)
Huzzah Feather ESP32 - Wifi and Bluetooth	32.50	2	65
40 pin header - 0.1" pitch single row breakable	0.85	1	0.85
Printed Circuit Board	25	1	25
Lithium Ion Battery - 3.7V 2000mAh	36.15 (w/ shipping)	1	36.15
1593ATS50 Screws - PCB screws	0	Pack of 50	0
Capacitors - Cap Film Rdl Polest 33000 pF 100V	2.71	1 (pack of 4)	2.71
STEVAL-MKI015V1 - Adapter board for LIS344ALH 3-axis accelerometer	100.72 (w/ shipping) + 24.05 duty	1	124.77
Elastic Strap	2.30/m	2m	4.60
Plastic Clip	3.2	1	3.20
Plastic Sliders	0.20	2	0.40
Plastic Enclosure	0	1	0
Total			262.68

Measurement and Testing Procedures

In our experiments, the acceleration signal that was produced by breathing motions was acquired with MEMS, LIS344ALH, 3-axis low power accelerometer with a sampling rate of 100Hz. The data is transmitted via Bluetooth Low Energy (BLE) to the ESP32 microcontroller, allowing for the equipment to run for a long time on our battery. The internal components of the device are shown in the figures below.

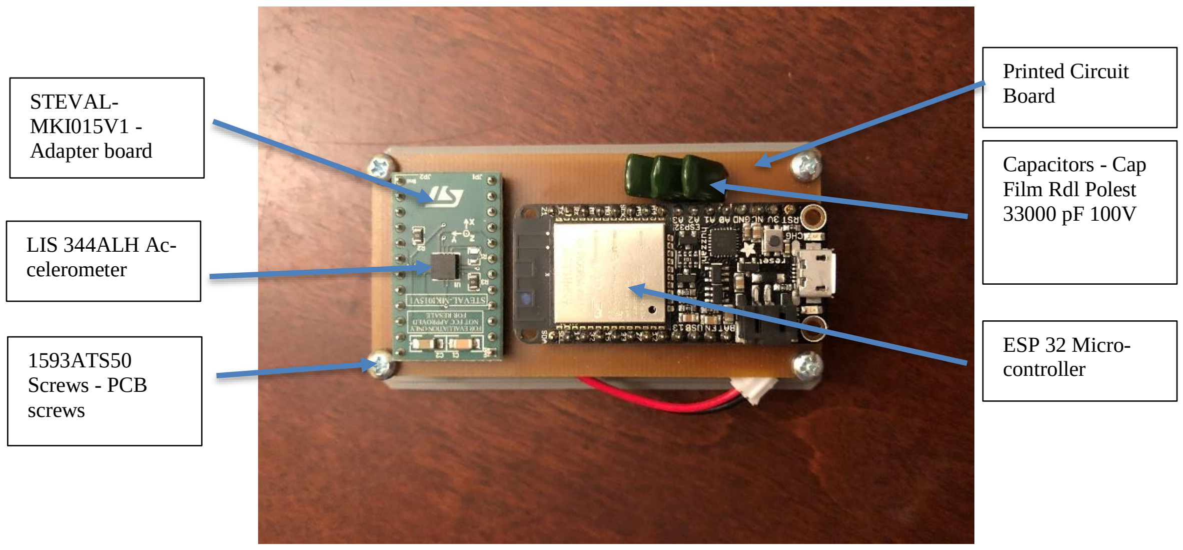

STEVAL-MKI015V1 - Adapter board
LIS 344ALH Accelerometer
1593ATS50 Screws - PCB screws
Printed Circuit Board
Capacitors - Cap Film Rdl Polest 33000 pF 100V
ESP 32 Microcontroller

Figure 10.0.1. Internal hardware components of the breathing monitor.

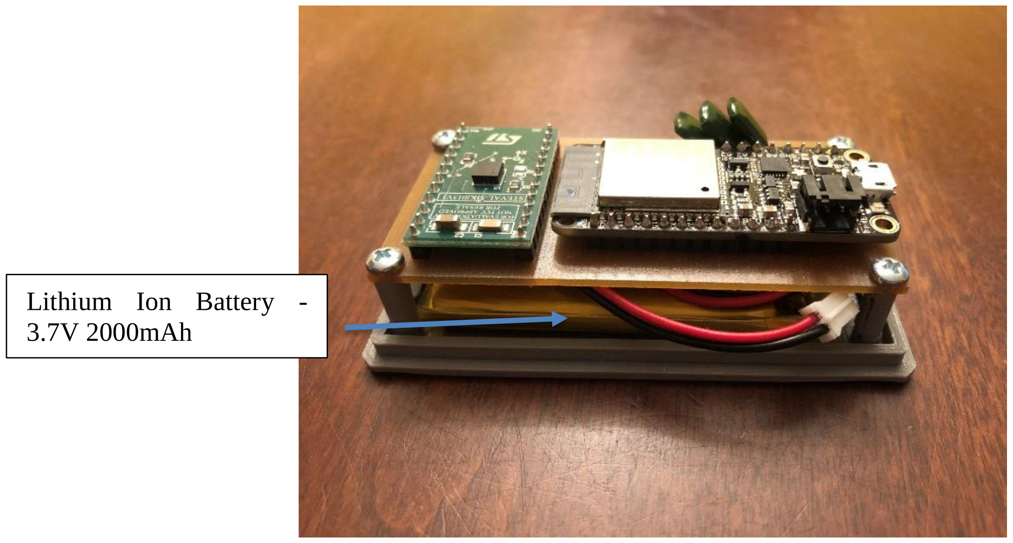


Figure 10.0.2. Internal hardware components of the breathing monitor (side view).

10.1 Test Setup

To verify our system, we performed several test measurements. Two subjects, A and B, were outfitted with our device. The participants in our measurement analysis included one male and one female. Prior to recording their breathing rate, both participants were instructed on how to perform the various breathing pathologies that we were looking to analyse. Each subject was recorded for approximately 90 minutes for the experimental trials. This ensured that our battery would be able to withstand that amount of time of usage, and that the data collected would be unbiased. The subjects were asked to perform various breathing rates including a normal breath (Eupnea), a slow breath (Bradypnea), and a quick breath (Tachypnea), each for 2 minutes. The subjects were then instructed to hold their breath (representing apnea), for a period of 15-18 s. The subjects were given a 3-minute rest interval.

The sensor was mounted onto each subject's abdomen, just below the sternum, and secured by a flexible elastic strap. The strap itself was easy to attach and comfortable to wear. In the trial sessions, each subject was instructed to maintain a seated position. The trial set up is shown in the figure below.

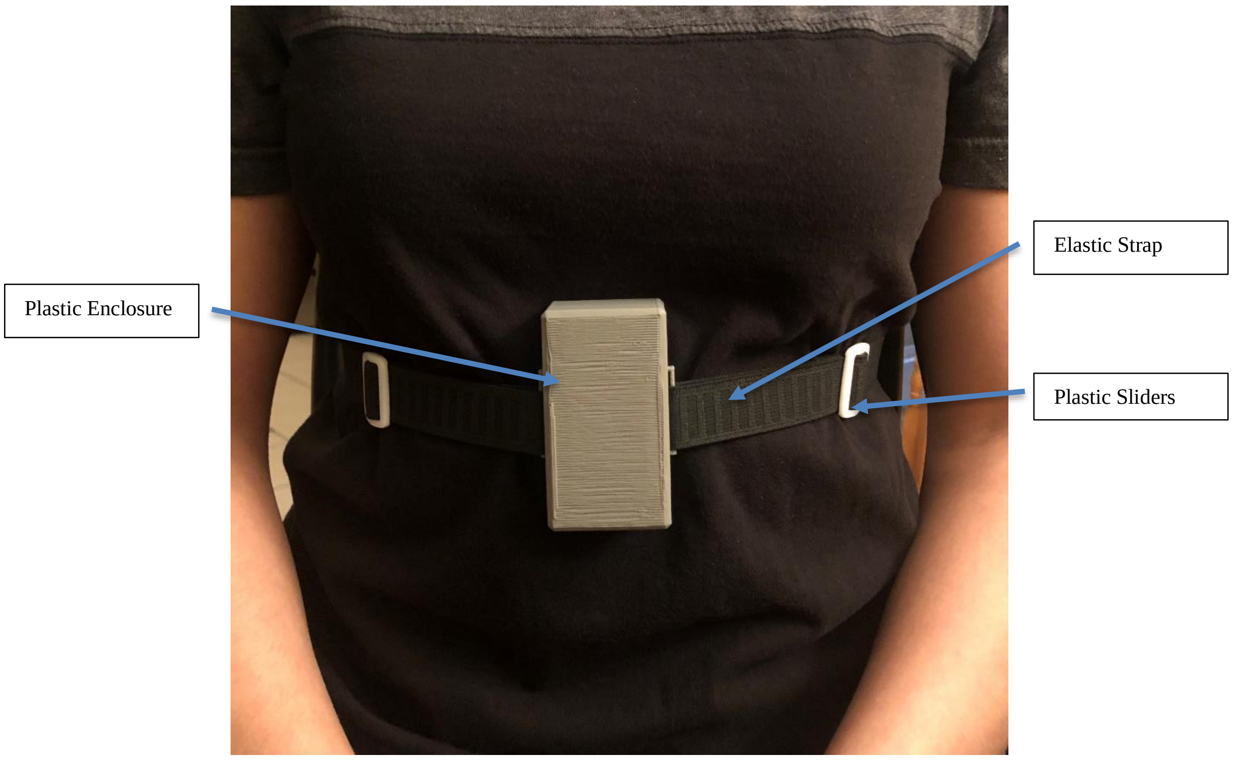


Figure 10.1.1. The wearable device mounted onto the subject's abdomen.

10.2 Measurement Results

During the test, the subjects were asked to perform 4 breathing rate patterns i.e., Normal, Bradypnea, Tachypnea and apnea. Bradypnea is a breathing pathology which is regular in rhythm but slower than normal in rate [4]. Contrasting to this, Tachypnea is a breathing pathology which is also regular in rhythm, but much faster than normal in rate [4]. Finally, apnea is a breathing pathology in which the breathing rate is absent, and the patient has a pause in breathing [4]. The figure below shows samples of all the patterns extracted from the accelerometer sensor.

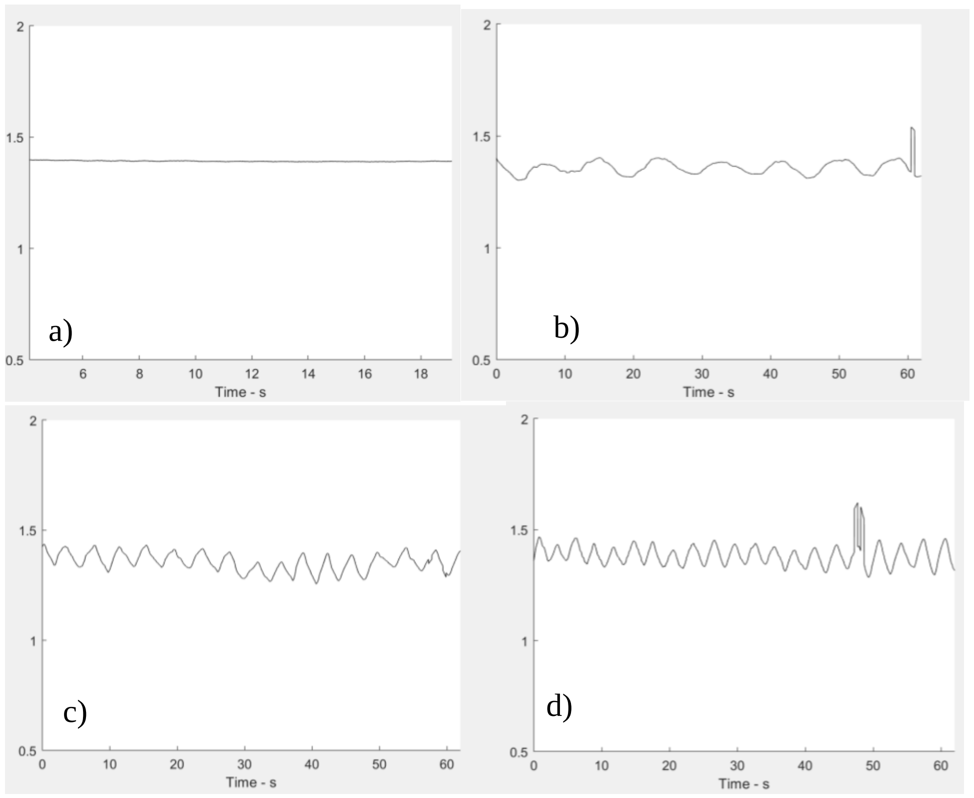


Figure 10.2.1. a) Apnea, b) Bradapnea, c) Normal, and d) Tachypnea breathing rates collected from the accelerometer sensor

In order to further test our algorithm, various window sizes were used in analysis. Window sizes of 15 seconds, 18 seconds, and 60 seconds were used and compared. The comparison between the various window sizes and the calculated breathing rates are shown in the appendix.

When comparing the various windows used for analysis, we chose to base our comparison on the percent error metric. This allowed for us to analyze the discrepancies between the true breathing rate counted over a 60 second time period and the observed breathing rate that was calculated using our algorithm. The results are summarized in the tables below.

Bradapnea	Observed breath/minute value (calculated from algorithm)	True breath/minute value	Pattern Correctly Classified	Percent Error
15 second window	8	7	Yes	14.2857%
18 second window	10	7	Yes	42.8571%
60 second window	6.77	7	Yes	3.2857%

Table 10.2.1. This table outlines the observed values, true values, and percent errors for various window sizes when analysing slow breathing (Bradapnea).

Tachypnea	Observed breath/minute value (calculated from algorithm)	True breath/minute value	Pattern Correctly Classified	Percent Error
15 second window	20	21	Yes	4.7619%
18 second window	23	21	Yes	9.5238%
60 second window	21	21	Yes	0%

Table 10.2.2. This table outlines the observed values, true values, and percent errors for various window sizes when analysing fast breathing (Tachypnea).

Eupnea	Observed breath/minute value (calculated from algorithm)	True breath/minute value	Pattern Correctly Classified	Percent Error
15 second window	12	16	Yes	25%
18 second window	16	16	Yes	0%
60 second window	16	16	Yes	0%

Table 10.2.3. This table outlines the observed values, true values, and percent errors for various window sizes when analysing normal breathing (Eupnea).

10.3 Post Analysis

Figures 12.2.1-12.2.8 show the results of our design. Our classification algorithm was able to classify 4 different breathing patterns based on the respiratory rate. The peak detection parameters were set to detect peaks using peak width and prominence. The minimum width and prominence ensured that high frequency components are ignored. A minimum distance parameter was also added due to several occasions where a peak would be marked twice. The patterns are defined by respiratory rate ranges. Either no breathing, slow breathing, fast breathing or normal breathing. We tested several different window sizes for the analysis portion of our design. The windows are non-overlapping and the analysis is performed when the array fills. We found little difference in accuracy of pattern classification when it came to different window sizes. However, the breathing rate is more accurate with larger window sizes. Ideally, a 60 second window would be used to best represent the breathing rate and pattern. In order to reduce the delay however, it was determined that the 18 second window would be used, as the breathing rates calculated from the 18 second window were comparable to those calculated from the 60 second window, resulting in the lowest percent error as show in *tables 12.2.1-12.2.3*.

The errors between the true values and the observed values could arise due to many different reasons, but most commonly due to limitations of the device used in measurement. Although the accelerometer that we used for our monitoring was highly sensitive, the accelerometer was not calibrated. Skipping the calibration step of our accelerometer was something that we opted out of doing due to the nature of our data acquisition. When acquiring data, the subject under analyzation is always still, in either a seated or lying down position. This means that the signal that we are interested in is always as a result of movement in the z-axis. Although the subject is at rest, it is possible that small movements could have been picked up by the accelerometer and mistakenly recorded as breathing motion, thus skewing the breaths per minute value.

Classification of the patterns only considers the rate. Therefore, it cannot detect patterns that have pauses or amplitude changes, such as Biot's, Kussmaul and Cheyn-Stoke's. The peak detection function used does have return values of the peak locations and their amplitudes. Using these values, we could further classify more patterns like those aforementioned.

Conclusions

The ability to detect breathing pathologies is crucial not only for a patient but also their caregiver. Since existing solutions are too inflexible for a patient, this project strives to provide a more practical real-time respiratory monitor. The proposed respiratory monitor detects the motion of the chest while breathing with a 3-axis accelerometer. The microcontroller connected to this accelerometer was able to send the data through Bluetooth Low Energy (BLE) to the MATLAB environment. From here, the information was put through various signal processing algorithms including smoothing, filtering, windowing, and peak detection. From the intervals between peaks, a breathing rate was be quantified, and a breathing pathology was diagnosed accordingly. These pathologies currently include apnea, tachypnea, and bradypnea.

In the initial project objectives, one of the major tasks stated was to develop a smartphone application that could be used to monitor the breathing the rate. Due to time constraints, this aspect of the project was unable to be completed. However, despite this, all other objectives that were stated in the design phase of this project were completed and fulfilled satisfactorily.

In researching and designing the algorithm for the breathing monitor, several problems were encountered. The first involved the use and implementation of an adaptive filter. Due to the nature of everyday life, humans rarely spend their day in a seated or lying down position [4]. This proves to be problematic as the movements associated with everyday life can influence the frequency content of respiration. The implementation of an adaptive filter [6,16,17], eliminates this issue as it can accommodate for various levels of activities. This is one of the limitations of our device, as we did not consider the implementation of an adaptive filter and will assume that the patient who wears the device will be in a seated, still position. This aspect of the project would need to be added to ensure that our device would be applicable in all scenarios and environments.

In addition to adaptive filtering, we encountered another design challenge with respect to classifying various breathing pathologies. The breathing pathologies that were classified are all related to the rate of breathing. There are several other breathing pathologies that exist that express themselves in different patterns as opposed to rates [4]. The proposed algorithm would not be able to differentiate between the various breathing patterns as it is focused more on the rate of breathing. This is a design flaw as our device would not be able to be used for all breathing monitoring, rather just for breathing rate monitoring. For future work on this project, we hope to be able to expand upon our current algorithm and develop a way to classify and monitor different breathing pathologies not associated with the rate of breathing.

Appendices

12.1 Schematics and Pin Layout

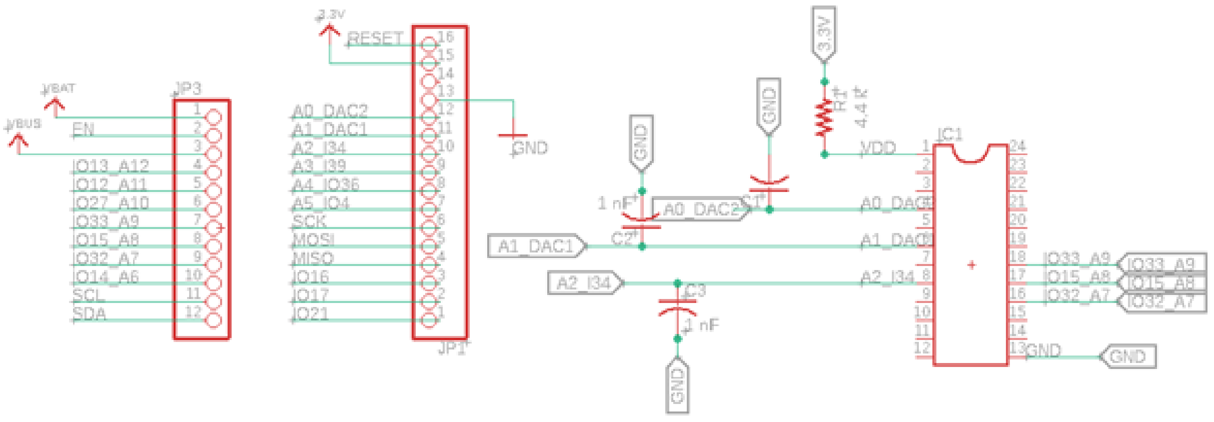


Figure 12.1.1 Circuit schematic with pin connections.

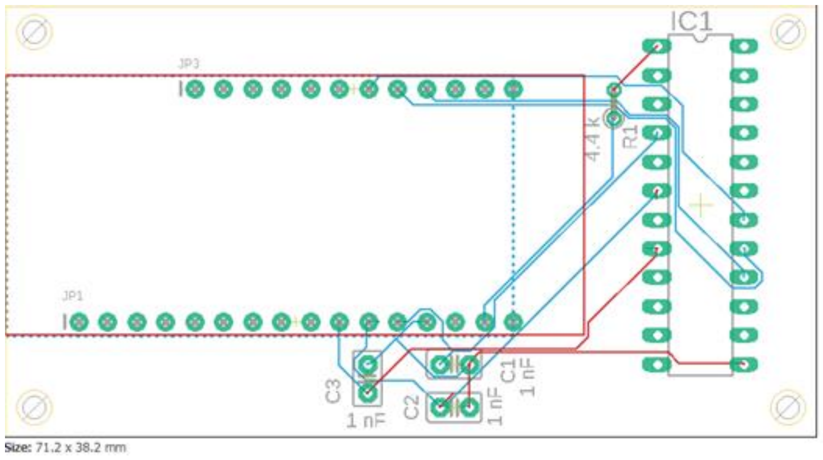


Figure 12.1.2. Eagle PCB design.

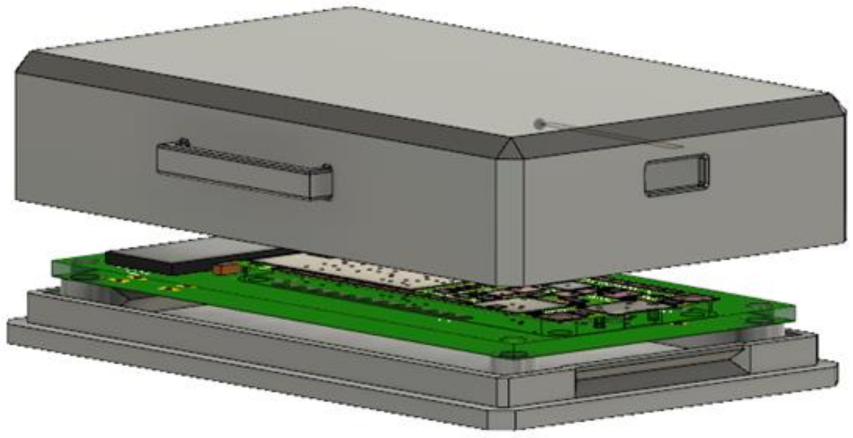


Figure 12.1.3. Case design with PCB.

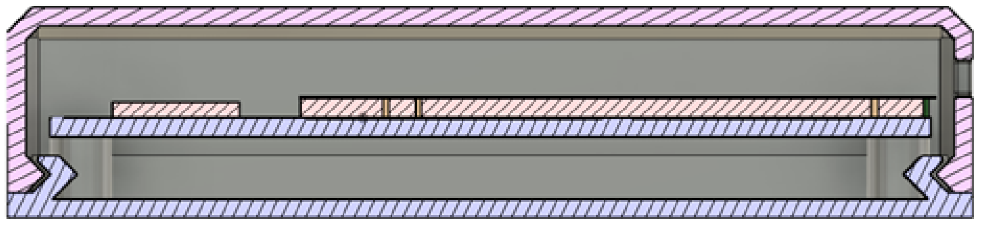
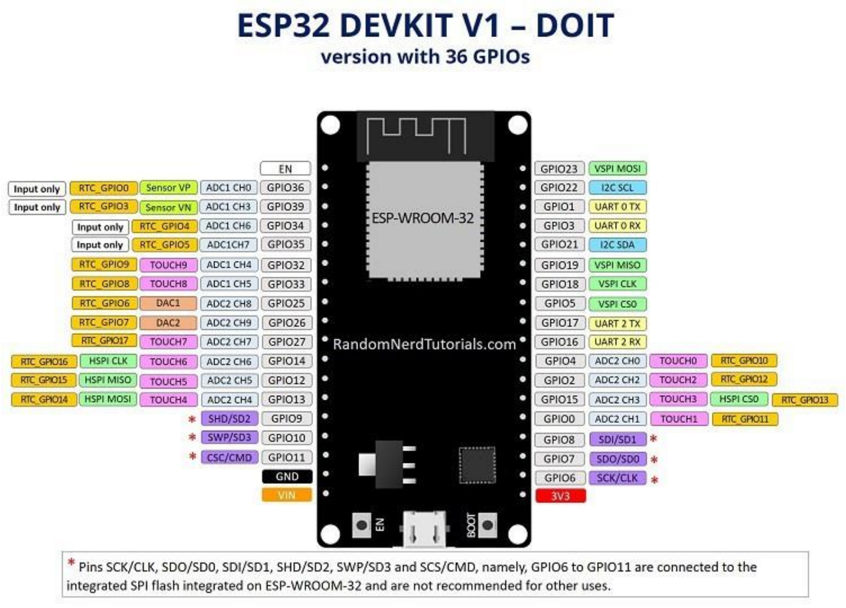


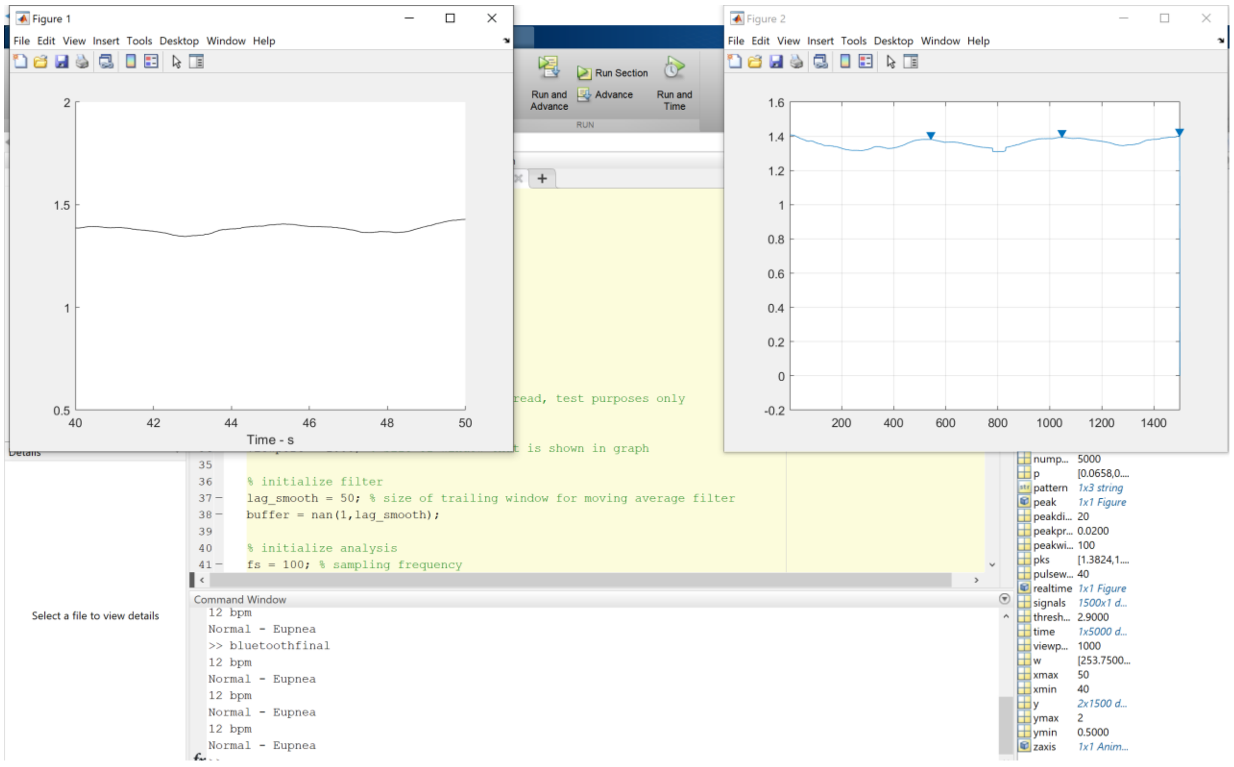
Figure 12.1.4. Cross-sectional view of case.


ESP32 DEVKIT V1 – DOIT
version with 36 GPIOs
ESP-WROOM-32
RandomNerdTutorials.com
EN
Input only | RTC_GPIO0 | Sensor VP | ADC1 CH0 | GPIO36
Input only | RTC_GPIO3 | Sensor VN | ADC1 CH3 | GPIO39
Input only | RTC_GPIO4 | ADC1 CH6 | GPIO34
Input only | RTC_GPIO5 | ADC1CH7 | GPIO35
RTC_GPIO9 | TOUCH9 | ADC1 CH4 | GPIO32
RTC_GPIO8 | TOUCH8 | ADC1 CH5 | GPIO33
RTC_GPIO6 | DAC1 | ADC2 CH8 | GPIO25
RTC_GPIO7 | DAC2 | ADC2 CH9 | GPIO26
RTC_GPIO17 | TOUCH7 | ADC2 CH7 | GPIO27
RTC_GPIO16 | HSPI CLK | TOUCH6 | ADC2 CH6 | GPIO14
RTC_GPIO15 | HSPI MISO | TOUCH5 | ADC2 CH5 | GPIO12
RTC_GPIO14 | HSPI MOSI | TOUCH4 | ADC2 CH4 | GPIO13
* SHD/SD2 | GPIO9
* SWP/SD3 | GPIO10
* CSC/CMD | GPIO11
GND
VIN
GPIO23 | VSPI MOSI
GPIO22 | I2C SCL
GPIO1 | UART 0 TX
GPIO3 | UART 0 RX
GPIO21 | I2C SDA
GPIO19 | VSPI MISO
GPIO18 | VSPI CLK
GPIO5 | VSPI CS0
GPIO17 | UART 2 TX
GPIO16 | UART 2 RX
GPIO4 | ADC2 CH0 | TOUCH0 | RTC_GPIO10
GPIO2 | ADC2 CH2 | TOUCH2 | RTC_GPIO12
GPIO15 | ADC2 CH3 | TOUCH3 | HSPI CS0 | RTC_GPIO13
GPIO0 | ADC2 CH1 | TOUCH1 | RTC_GPIO11
GPIO8 | SDI/SD1 *
GPIO7 | SDO/SD0 *
GPIO6 | SCK/CLK *
3V3
EN
BOOT

* Pins SCK/CLK, SDO/SD0, SDI/SD1, SHD/SD2, SWP/SD3 and SCS/CMD, namely, GPIO6 to GPIO11 are connected to the integrated SPI flash integrated on ESP-WROOM-32 and are not recommended for other uses.

Figure 12.1.5. ESP32 Microcontroller pin layout.

12.2 Results



```
read, test purposes only

t is shown in graph

% initialize filter
lag_smooth = 50; % size of trailing window for moving average filter
buffer = nan(1,lag_smooth);

% initialize analysis
fs = 100; % sampling frequency
```

Command Window

```
12 bpm
Normal - Eupnea
>> bluetoothfinal
12 bpm
Normal - Eupnea
12 bpm
Normal - Eupnea
12 bpm
Normal - Eupnea
```

nump...	5000
p	[0.0658,0....
pattern	1x3 string
peak	1x1 Figure
peakdi...	20
peakpr...	0.0200
peakwi...	100
pks	[1.3824,1....
pulsew...	40
realtime	1x1 Figure
signals	1500x1 d...
thresh...	2.9000
time	1x5000 d...
viewp...	1000
w	[253.7500...
xmax	50
xmin	40
y	2x1500 d...
ymax	2
ymin	0.5000
zaxis	1x1 Anim...

Figure 12.2.1. Normal breathing classification with a window size of 15 seconds.

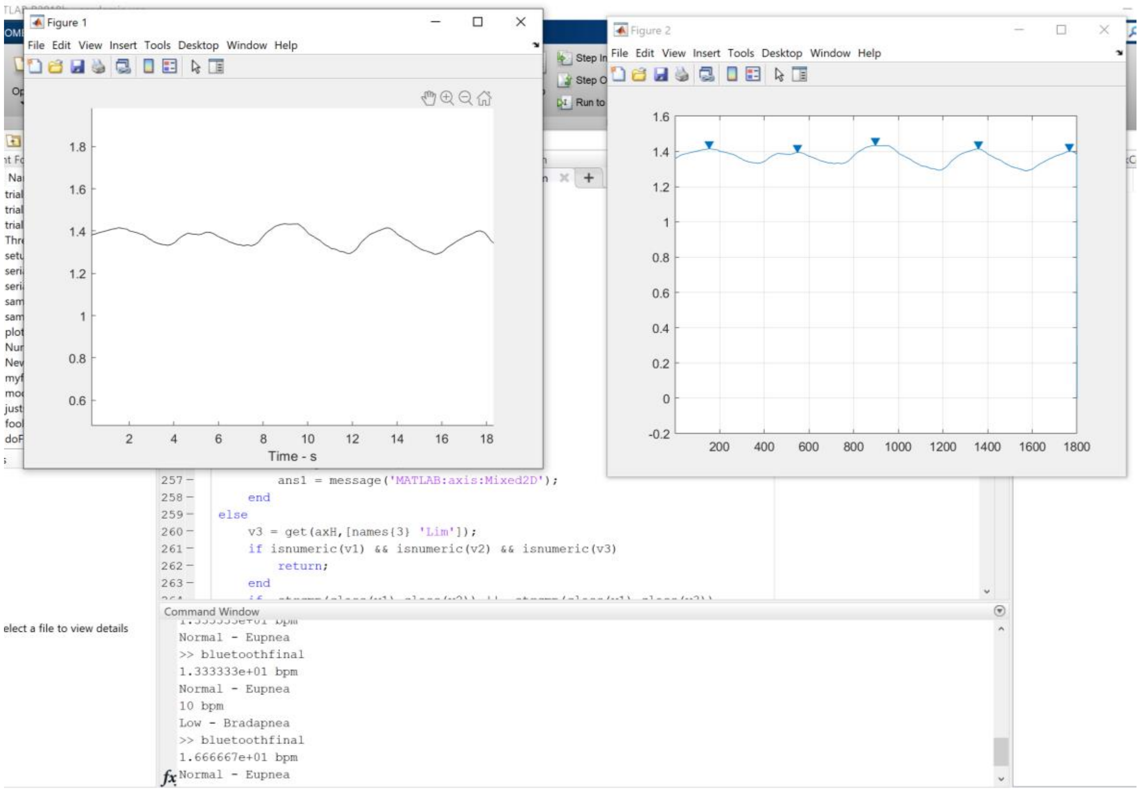
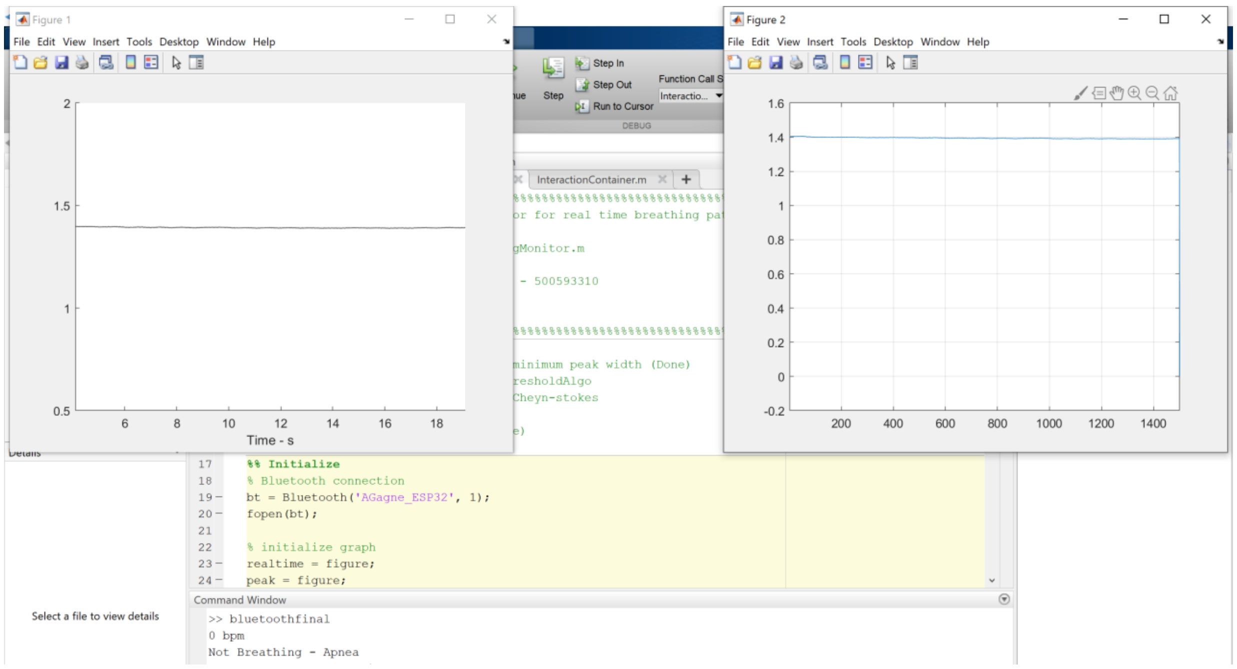


Figure 1
File Edit View Insert Tools Desktop Window Help
Time - s
Figure 2
File Edit View Insert Tools Desktop Window Help
Step In
Step Out
Run to
Command Window
Select a file to view details

```
        ans1 = message('MATLAB:axis:Mixed2D');
    end
else
    v3 = get(axH,[names{3} 'Lim']);
    if isnumeric(v1) && isnumeric(v2) && isnumeric(v3)
        return;
    end
```

```
Normal - Eupnea
>> bluetoothfinal
1.333333e+01 bpm
Normal - Eupnea
10 bpm
Low - Bradapnea
>> bluetoothfinal
1.666667e+01 bpm
Normal - Eupnea
```

Figure 12.2.2. Normal breathing classification with a window size of 18 seconds.

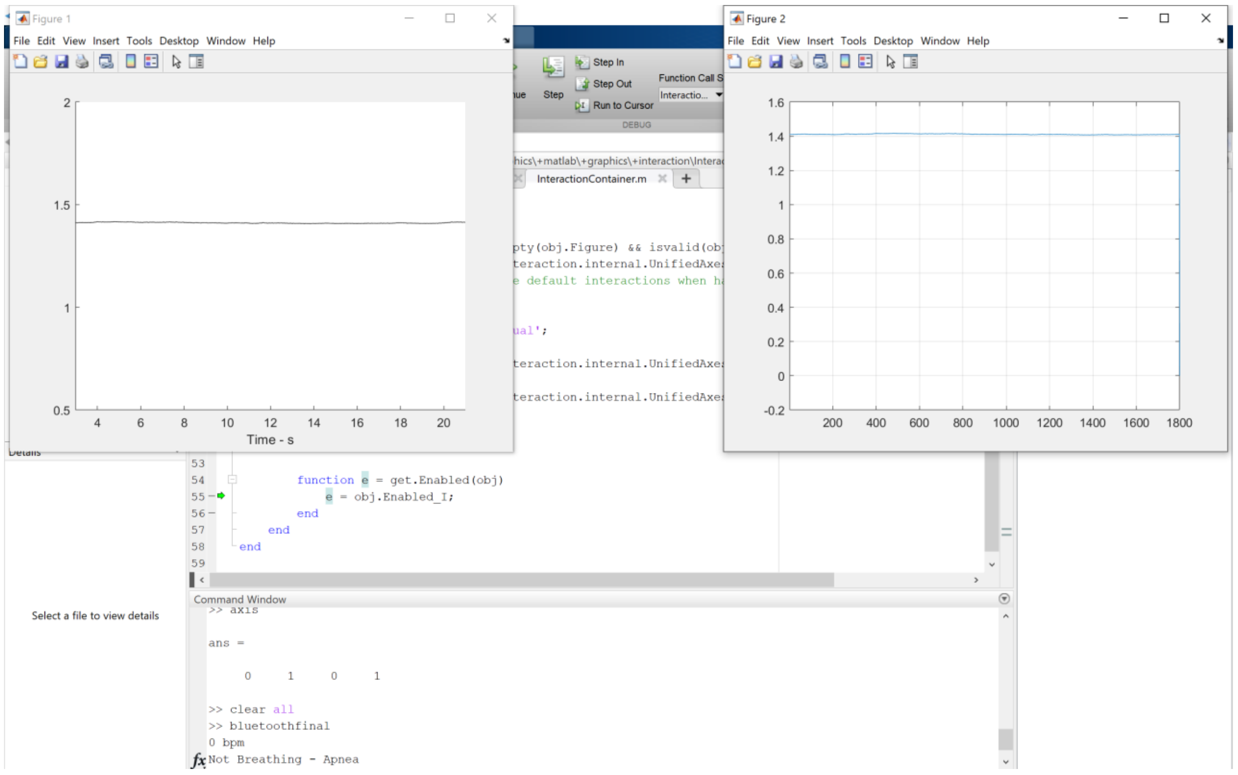


```
or for real time breathing pa
gMonitor.m
 - 500593310
minimum peak width (Done)
resholdAlgo
Cheyn-stokes
```

```
%% Initialize
% Bluetooth connection
bt = Bluetooth('AGagne_ESP32', 1);
fopen(bt);

% initialize graph
realtime = figure;
peak = figure;
```

```
>> bluetoothfinal
0 bpm
Not Breathing - Apnea
```

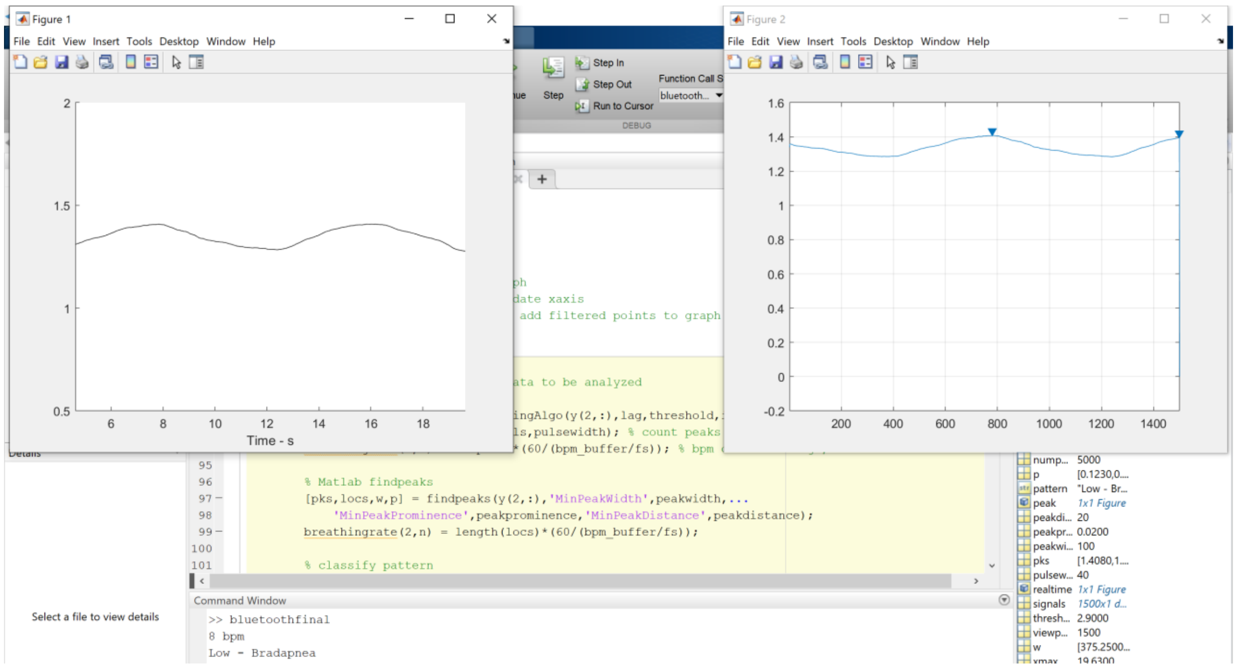
Figure 12.2.3. Apnea breathing classification with a window size of 15 seconds.



```
pty(obj.Figure) && isvalid(ob
teraction.internal.UnifiedAxe
e default interactions when h
ual';
teraction.internal.UnifiedAxe
teraction.internal.UnifiedAxe

        function e = get.Enabled(obj)
            e = obj.Enabled_I;
        end
    end
end
```

```
>> axis

ans =

     0     1     0     1

>> clear all
>> bluetoothfinal
0 bpm
Not Breathing - Apnea
```

Figure 12.2.4. Apnea breathing classification with a window size of 18 seconds.

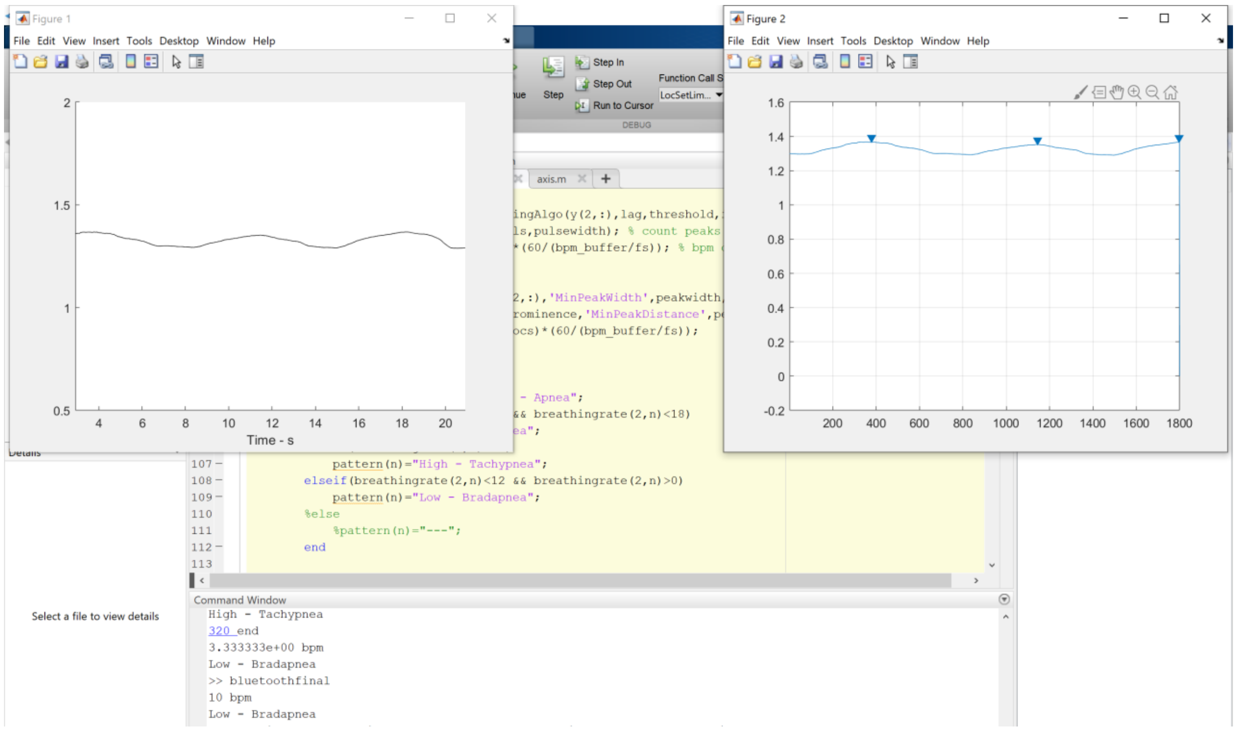


```
ph
date xaxis
 add filtered points to graph
ata to be analyzed
ingAlgo(y(2,:),lag,threshold,
ls,pulsewidth); % count peaks
*(60/(bpm_buffer/fs)); % bpm

        % Matlab findpeaks
        [pks,locs,w,p] = findpeaks(y(2,:),'MinPeakWidth',peakwidth,...
            'MinPeakProminence',peakprominence,'MinPeakDistance',peakdistance);
        breathingrate(2,n) = length(locs)*(60/(bpm_buffer/fs));

        % classify pattern
```

```
>> bluetoothfinal
8 bpm
Low - Bradapnea
```

nump...	5000
p	[0.1230,0....
pattern	"Low - Br...
peak	1x1 Figure
peakdi...	20
peakpr...	0.0200
peakwi...	100
pks	[1.4080,1....
pulsew...	40
realtime	1x1 Figure
signals	1500x1 d...
thresh...	2.9000
viewp...	1500
w	[375.2500...
xmax	19.6300

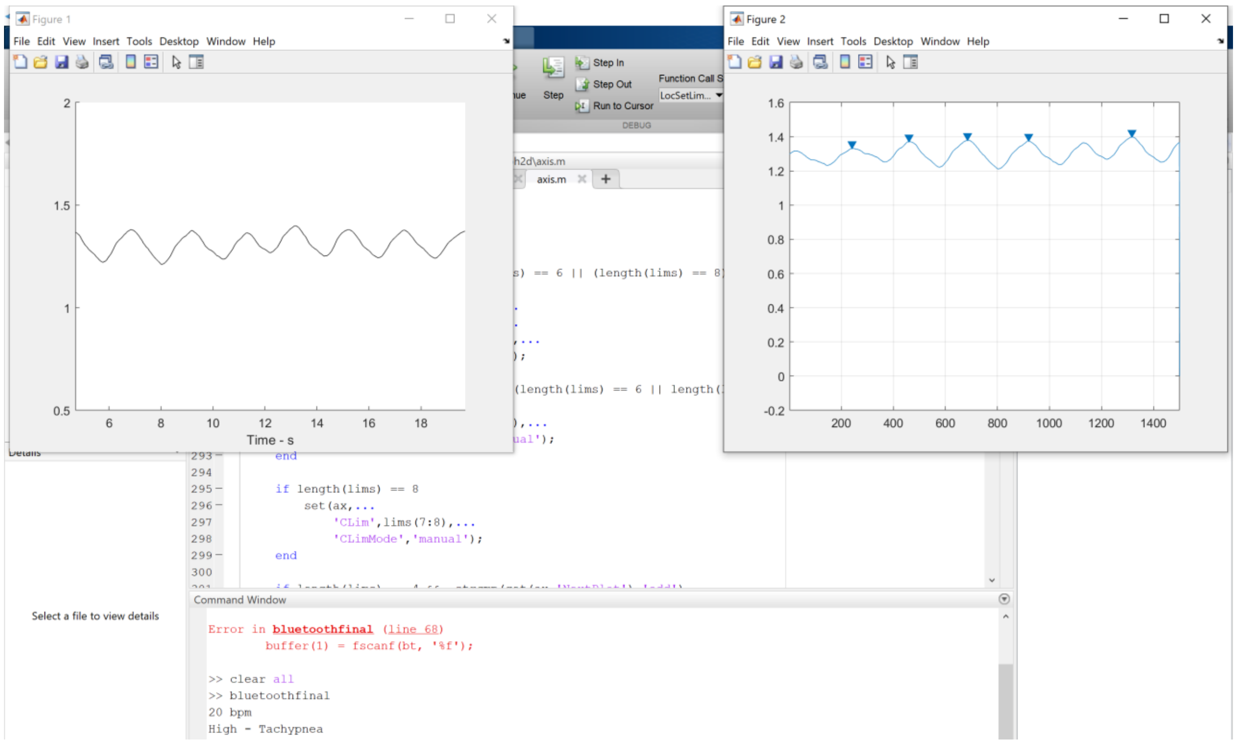
Figure 12.2.5. Bradapnea breathing classification with a window size of 15 seconds.



```
ingAlgo(y(2,:),lag,threshold,
ls,pulsewidth); % count peaks
*(60/(bpm_buffer/fs)); % bpm

2,:),'MinPeakWidth',peakwidth,
rominence,'MinPeakDistance',p
ocs)*(60/(bpm_buffer/fs));

 - Apnea";
&& breathingrate(2,n)<18)
ea";
        pattern(n)="High - Tachypnea";
    elseif(breathingrate(2,n)<12 && breathingrate(2,n)>0)
        pattern(n)="Low - Bradapnea";
    %else
        %pattern(n)="---";
    end
```

```
High - Tachypnea
320 end
3.333333e+00 bpm
Low - Bradapnea
>> bluetoothfinal
10 bpm
Low - Bradapnea
```

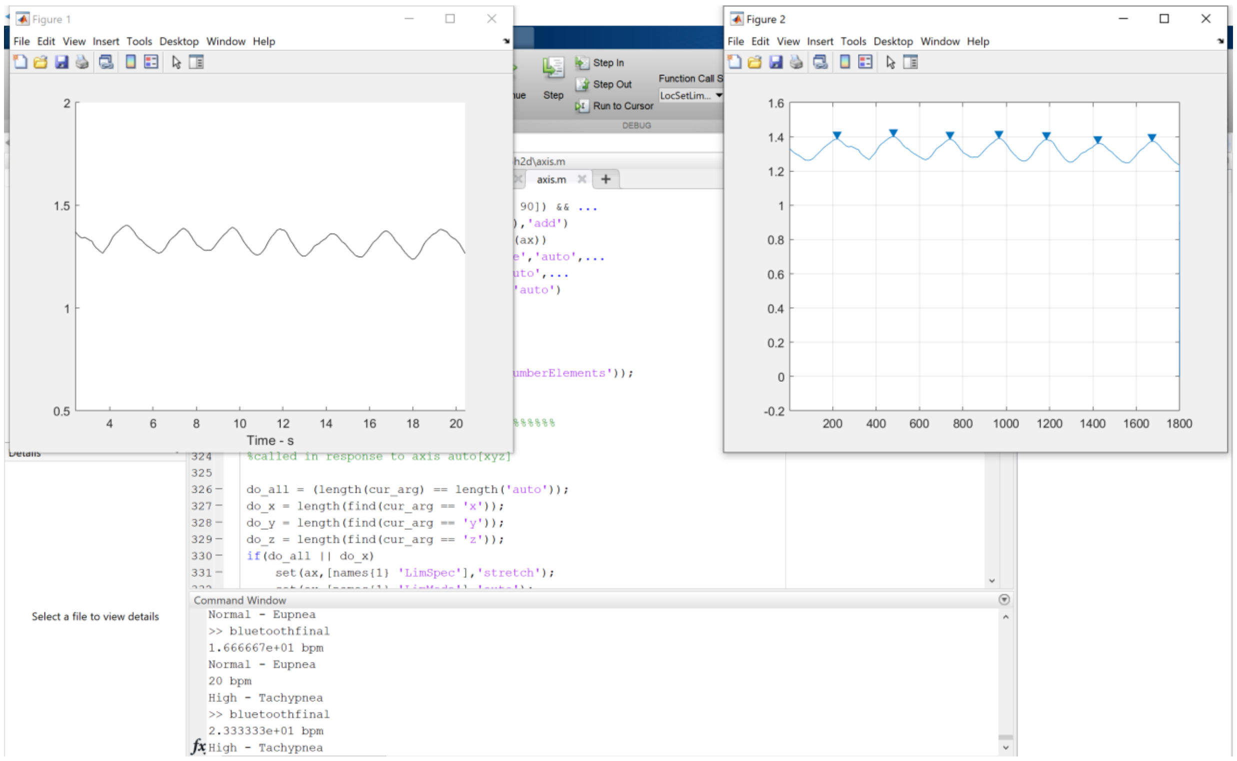
Figure 12.2.6. Bradapnea breathing classification with a window size of 18 seconds.



```
s) == 6 || (length(lims) == 8

,...
);

(length(lims) == 6 || length(

),...
ual');
    end

    if length(lims) == 8
        set(ax,...
            'CLim',lims(7:8),...
            'CLimMode','manual');
    end
```

```
Error in bluetoothfinal (line 68)
        buffer(1) = fscanf(bt, '%f');

>> clear all
>> bluetoothfinal
20 bpm
High - Tachypnea
```

Figure 12.2.7. Tachypnea breathing classification with a window size of 15 seconds.



```
%called in response to axis auto[xyz]

do_all = (length(cur_arg) == length('auto'));
do_x = length(find(cur_arg == 'x'));
do_y = length(find(cur_arg == 'y'));
do_z = length(find(cur_arg == 'z'));
if(do_all || do_x)
    set(ax,[names{1} 'LimSpec'],'stretch');
```

```
Normal - Eupnea
>> bluetoothfinal
1.666667e+01 bpm
Normal - Eupnea
20 bpm
High - Tachypnea
>> bluetoothfinal
2.333333e+01 bpm
High - Tachypnea
```

Figure 12.2.8. Tachypnea breathing classification with a window size of 18s.

12.3 Arduino code implemented for ESP32

```
#include "BluetoothSerial.h"
#define zaxis A2
BluetoothSerial SerialBT;

void setup() {
SerialBT.begin("AGagne_ESP32"); //Bluetooth device name
            }

void loop() {
if (SerialBT.available()>0) {
 char read = SerialBT.read();
                switch(read){
                 case 'E':
                  start();
                  break;
                }
              }
            }

void start(){
            while(1){
                // SerialBT.print('s'); // ANDROID APP
                // SerialBT.print(floatMap(analogRead(zaxis),0,4095,0,3.3),2);
```

```
    SerialBT.println(floatMap(analogRead(A2),0,4095,0,3.3),2); // Matlab

    delay(10);

    if(SerialBT.available()>0){
      if(SerialBT.read()=='Q'){
        return;
      }
    }
  } // while
} // start

float floatMap(float x, float inMin, float inMax, float outMin, float outMax) {
  return (x-inMin)*(outMax-outMin)/(inMax-inMin)+outMin;
}
```

12.4 MATLAB code implemented for signal processing and analysis

```
%% Initialize
% Bluetooth connection
bt = Bluetooth('AGagne_ESP32', 1);
fopen(bt);

% initialize graph
realtime = figure;
peak = figure;
movegui(realtime,'northwest');
movegui(peak,'northeast');
figure(realtime);
zaxis = animatedline();
xlabel('Time - s');
j = 1;
numpoints = 18000; % number of points read, test purposes only
ymin = 0.5;
ymax = 2;
viewport = 1500; % size of window that is shown in graph

% initialize filter
lag_smooth = 50; % size of trailing window for moving average filter
buffer = nan(1,lag_smooth);

% initialize analysis
fs = 100; % sampling frequency
bpm_buffer = 1500; % amount of data analyzed at a time
y = zeros(2,bpm_buffer);
m = 1;
n = 1;

% smoothing zscore parameters
lag = 50; % how much your data will be smoothed and how adaptive the
          % algorithm is to changes in the long-term average of the data
threshold = 2.9; % number of standard deviations from the moving mean
                 % above which the algorithm will classify a new datapoint
                 % as being a signal
influence = 0; % influence of signals on the algorithm's detection threshold
```

```
pulsewidth = 40; % minimum width of 'signals' that is considered a peak

% matlab 'findpeaks' parameters - width,height,prominence,threshold,distance
peakwidth = 100; % MinPeakWidth
peakprominence = 0.02; % MinPeakProminence
peakdistance = 15; % MinPeakDistance

fprintf(bt,'%s','E'); % send start bit to arduino

for k = 1:numpoints
    %% Filter Data
    % Smoothing Filter
    while(isnan(buffer(lag_smooth))) % if buffer is not full, fill it
        buffer = circshift(buffer,1);
        buffer(1) = fscanf(bt, '%f');
    end
    filtered = mean(buffer); % calculate moving average
    buffer = circshift(buffer,1); % shift points
    buffer(1) = fscanf(bt, '%f'); % add datapoint to buffer

    %% Display Data
    % set scrolling axis
    if(j>viewport) % moving viewport axis limits
        xmin=(j-viewport)/fs;
        xmax=j/fs;
    else % initial viewport
        xmin=0;
        xmax=viewport/fs;
    end

    % update axis and add data to graph
    axis([xmin xmax ymin ymax]); % update xaxis
    addpoints(zaxis,j/fs,filtered); % add filtered points to graph
    drawnow limitrate % update graph

    %% Analysis
    if(m == bpm_buffer) % amount of data to be analyzed
        % Smoothing zscore
        [signals,avg,dev] = ThresholdingAlgo(y(2,:),lag,threshold,influence); % send y to threshold algo
        [numpeaks] = countPeaks(signals,pulsewidth); % count peaks
        breathingrate(1,n) = numpeaks*(60/(bpm_buffer/fs)); % bpm of zscore algo, convert s to min

        % Matlab findpeaks
        [pks,locs,w,p] = findpeaks(y(2,:),'MinPeakWidth',peakwidth,...
            'MinPeakProminence',peakprominence,'MinPeakDistance',peakdistance);
        breathingrate(2,n) = length(locs)*(60/(bpm_buffer/fs));

        % classify pattern
        if(breathingrate(2,n)<1)
            pattern(n)="Not Breathing - Apnea";
        elseif(breathingrate(2,n)>11 && breathingrate(2,n)<18)
            pattern(n)="Normal - Eupnea";
        elseif(breathingrate(2,n)>18)
            pattern(n)="High - Tachypnea";
        elseif(breathingrate(2,n)<12 && breathingrate(2,n)>0)
            pattern(n)="Low - Bradapnea";
        %else
            %pattern(n)="---";
        end

        data{1,n} = y; % test purposes only, store data

        % display results
        figure(peak);
        findpeaks(y(2,:),'MinPeakWidth',peakwidth,...
            'MinPeakProminence',peakprominence,'MinPeakDistance',peakdistance);
        fprintf('%d bpm\n',breathingrate(2,n));
        fprintf('%s\n',pattern(n));

        figure(realtime);
        n=n+1;
        m=1;
    else
        y(1,m) = j/fs; % time
        y(2,m) = filtered; % add point to buffer to be analyzed
        m=m+1;
    end

    j=j+1;

end

fprintf(bt,'%s','Q'); % send stop bit to arduino

[time,breath] = getpoints(zaxis); % test purposes, store data

fclose(bt);
clear('bt');

function [signals,avgFilter,stdFilter] = ThresholdingAlgo(y,lag,threshold,influence)
% Initialise signal results
signals = zeros(length(y),1);
% Initialise filtered series
filteredY = y(1:lag+1);
% Initialise filters
avgFilter(lag+1,1) = mean(y(1:lag+1));
stdFilter(lag+1,1) = std(y(1:lag+1));
% Loop over all datapoints y(lag+2),...,y(t)
for i=lag+2:length(y)
    % If new value is a specified number of deviations away
    if abs(y(i)-avgFilter(i-1)) > threshold*stdFilter(i-1)
        if y(i) > avgFilter(i-1)
            % Positive signal
            signals(i) = 1;
        else
            % Negative signal
            signals(i) = -1;
        end
        % Make influence lower
        filteredY(i) = influence*y(i)+(1-influence)*filteredY(i-1);
    else
        % No signal
        signals(i) = 0;
        filteredY(i) = y(i);
    end
    % Adjust the filters
    avgFilter(i) = mean(filteredY(i-lag:i));
    stdFilter(i) = std(filteredY(i-lag:i));
end
% Done, now return results
end

function [peaks] = countPeaks(signals,pulsewidth)
% Count falling edges in signals array
% Peak must have minimum width
peaks = 0;
range = ones(pulsewidth,1);
for i = 1:(length(signals)-pulsewidth-1)
    if isequal(range,signals(i:(i+pulsewidth-1))) && ...
            (signals(i+pulsewidth)==0||signals(i+pulsewidth)==-1)
        peaks = peaks + 1;
    end
end

end
```

References

[1] S. Braun, *Clinical Methods: The History, Physical, and Laboratory Examinations.*, 3rd ed. Boston: Butterworths, 1990.
[2] K. Barrett, S. Barman, H. Brooks, J. Yuan and W. Ganong, *Ganong's review of medical physiology*. .
[3] J. Dempsey, S. Veasey, B. Morgan and C. O'Donnell, "Pathophysiology of Sleep Apnea", *Physiological Reviews*, vol. 90, no. 1, pp. 47-112, 2010.
[4] W. Dorland, *Dorland's illustrated medical dictionary*. Philadelphia, Pa.: Saunders/Elsevier, 2012.
[5] A. Al-Naji and J. Chahl, "Remote respiratory monitoring system based on developing motion magnification technique", *Biomedical Signal Processing and Control*, vol. 29, pp. 1-10, 2016.
[6] J. Vertens, F. Fischer, C. Heyde, F. Hoeflinger, R. Zhang, L. Reindl and A. Gollhofer, "Measuring respiration and heart rate using two acceleration sensors on a fully embedded platform," *icSPORTS*, vol. 3, pp. 15-23, 2015. [Online]. Available: DOI: 10.5220/0005604000150023.
[7] C. Landon, and FAAP, FCCP, "Respiratory monitoring: Advantages of inductive plethysmography over impedance pneumography," 2018. [E-Book]. Available: https://www.researchgate.net/publication/237682810.
[8] M. Krehel, M. Schmid, R.M. Rossi, L.F. Boesel, G.L. Bona, and L.J. Scherer, "An optical fibre-based sensor for respiratory monitoring," *Sensors*, vol. 14, no. 7, pp. 13088-101, Jul. 2014. [Online]. Available: doi:10.3390/s140713088.
[9] P. Arlotto, M. Grimaldi, R. Naeck, and J.M. Ginoux, "An ultrasonic contactless sensor for breathing monitoring," *Sensors*, vol. 14, no. 8, pp. 15371-86, Aug. 2014. [Online]. Available: doi: 10.3390/s140815371.
[10] A.R. Fekr, M. Janidarmian, K. Radecka, and Z. Zilic, "Development of a remote monitoring system for respiratory analysis," *Internet of Things. User-Centric IoT. IoT360 2014. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, vol. 150, pp. 193-202, June 2015. [Online]. Available: https://doi.org/10.1007/978-3-319-19656-5_28.
[11] F.A. Cespedes, "Respiratory monitoring system based on the thoracic expansion measurement," M.S. Theses, Elec. and Comp. Eng., Univ. of South Fl., Jan. 2012. [Online]. Available: https://scholarcommons.usf.edu/etd/3953/.
[12] Z. Cao, R. Zhu, R. Que, "A Wireless Portable System With Microsensors for Monitoring Respiratory Diseases", *IEEE Transaction on Biomedical Engineering*, vol. 59, *(11)*, pp. 3110-3116, 2012.
[13] R.K Kodali and K.S Mahesh, "Smart emergency response system", *TENCON 2017-2017 IEEE Region 10 Conference*, pp 712-717, 2017.
[14] I. Allafi and T. Iqbal, "Design and implementation of a low cost web server using ESP32 for real-time photovoltaic system monitoring", *2017 IEEE Electrical Power and Energy Conference*, pp.1-5, 2017.
[15] Espressif Systems, "ESP32-WROOM-32", ESP32 datasheet, 2018.

[16] A. R. Fekr et al, "A medical cloud-based platform for respiration rate measurement and hierarchical classification of breath disorders," Sensors (Basel, Switzerland), vol. 14, (6), pp. 11204-11224, 2014.
[17] P. D. Hung et al, "Estimation of respiratory waveform using an accelerometer," in 2008, Available: DOI: 10.1109/ISBI.2008.4541291.
[18] C.C. Yang, and Y.L. Hsu, "A review of accelerometry-based wearable motion detectors for physical activity monitoring," Sensors, vol. 10, pp. 7772-7788, Aug. 2010. [Online]. Available: doi:10.3390/s100807772.
[19] STMicroelectronics, "MEMS inertial sensor high performance 3-axis ±2/±6g ultra compact linear accelerometer," LIS344ALH datasheet, Jan. 2008 [Revised Apr. 2008].
[20] Adafruit, Adafruit HUZZAH32 - ESP32 Feather datasheet, [Revised Oct. 2018].
[21] STMicroelectronics, "LIS344ALH adapter board for a standard DIL24 socket," STEVAL-MKI015V1 datasheet, Oct. 2010 [Revised Aug. 2017].
[22]S. Liang, *The Java Native interface*. Reading, MA: Addison-Wesley, 1999.